## Module Requirements: Calibration

The calibration is an important part of ensuring the accuracy of the entire system, and is required to be able to accurately adjust the way the program converts the raw voltage from the scales strain gauges into numerically represented mass.

### Functional Requirements

The main function requirement of calibration was accuracy of results, since they would directly relate to the accuracy of the entire system.

#### Inputs

The calibration needs to receive voltages and masses as input, in order to create a conversion equation. Unfortunately, these inputs are prone to being highly inaccurate. The strain gauges supplying the voltage are greatly affected by vibrations and oscillations, and the input masses themselves can be inaccurate. Additionally, a mass entered by the user can be inaccurate, either because of their own error in weighing, or if not enough significant figures are entered (particularly a problem when dealing with ounces).

#### Process

The internal process of the calibration module has to be as accurate as possible. Accuracy to many significant figures is preferred over integers.

Ideally, the module’s calculations should be able to produce an accurate calibration, which will produce a weight value with a maximum of ±1 g error, and it should be able to do so for any mass from 1 g to 1 kg.

#### Outputs

The output of the calibration module should ideally be as accurate as possible. Floats are preferred over integers, because this will enhance the accuracy of other parts of the program. Additionally, it must be able to output in both ounces and grams, no matter the input unit.

### Non-Functional Requirements

Since the calibration module is mostly concerned with supplying accurate data to other parts of the system, the non-functional requirements are less important than the functional. However, there are still several aspects of non-functional requirements, which were considered in some depth during design of the module.

#### Performance

The computational complexity of the calibration module was a minor consideration because it would only be performed occasionally, and with not in real-time.

#### Interfaces

Simplicity for the user was an important issue. Whilst it would be possible to get a quite accurate system by creating a table of all possible voltage and mass combinations up to 1 kg, this would be an extremely difficult system to use. Ideally, the system should be able to produce a decent calibration with just two distinct voltage and mass readings.

#### Design Constraints

The limitations of the C programming language and the PIC microcontroller meant that extremely complicated calibrations were out of the question. It is difficult to perform complicated calculations, and too many calculations lead to a variety of errors, including rounding errors. The best solution to the calibration problem would likely be an n-degree polynomial line of best fit, however doing this using the tools available is quite impractical.

## Conceptual Design: Calibration

Early tests with the hardware indicated that the relationship between input voltage and mass was fairly linear. An oscilloscope was used to obtain voltages for a variety of test masses. Whilst the complexity of the beam meant that true linearity was unlikely, it was sufficiently so that it was deemed acceptable to use linear fit for calibration.

To improve accuracy and speed of calculation, it was decided that the calibration and weigh methods should go directly from the raw A/D value to the mass, rather than converting the voltage to a value between 0 and 5 V first. Thus, the following graph was produced using the same data as above, but inverting the axes and using voltages as if they were the 10 bit A/D values which the microcontroller would see.

Initial prototypes of the calibration module had a very simple method, based on the above research. Using integer arithmetic for speed, the module created a linear line of best fit using a simple equation, involving calculating gradient and y‑intercept from the two data points. However, this was quickly found to be highly inaccurate for the following reasons:

* Integer arithmetic does not provide sufficient accuracy and is prone to rounding errors
* The simple equation did not fully utilize the potential accuracy available
* Two data points were a poor representation of the entire voltage-mass relationship

The last point was deemed too difficult to fix, as a more complicated fit line would be both expensive both in computational and memory terms, and it would be highly susceptible to errors in the microcontroller C environment.

After integration of some of the modules of the system, it was found that computational complexity was far less important than originally thought. Thus, it was simple to change the calibration module to use float arithmetic.

Whilst the basic method of using a straight linear fit between two points was acceptable, it was possible to improve the accuracy of the results. By multiplying the 10 bit A/D voltage values by 64, they took up a full 16 bit integer. Whilst not useful in itself, this meant that when a (fractional) gradient is applied, a larger number remains, which is more accurate. The basic weigh function was modified to:

The gradient was used as an inverse because the value would always fall in the range (0,1). By calculating and storing the inverse a greater degree of accuracy could be achieved.

Assuming two data points, at 0 mass with some voltage, and another known mass with some other voltage, the gradient-1 and offset variables can be isolated.

It is now quite easy to use these equations to calculate *g* and *f*.  
  
Another consideration was the need to produce the results in both ounces and grams. It was decided that it was impractical to perform the whole process with different mass units, and thus the most effective method would be to convert the result. It was decided that a unit conversion function would be used. To convert the g value, it would need to be divided by the conversion factor, whilst the *f* value would need to be multiplied.

Once all of these problems were solved, it was simple to implement the module in software.

### Assumptions Made

It is assumed that all the values used during calibration are roughly accurate. Specifically, it is assume that the known mass is entered properly. An incorrect input will likely produce an extremely incorrect output.

### Constraints on Performance

The inaccuracy of the scales is a serious problem for the calibration. Even slight fluctuations in the voltage caused by small oscillations can throw off the result significantly. If a mass smaller than the calibration mass is weighed, it is likely to be more accurate that if a larger mass is weighed, because the interpolation is more accurate than extrapolation.

## Software Design: Calibration

Interface  
  
The calibration methods are part of the Scales module, but are accessed primarily by an external module containing the user interface, in this case the States module. The calibration can be performed automatically by calling the calibrate function. The following values are required as parameters:

* The A/D voltage when the scales are empty
* The A/D voltage when the scales contain a known mass
* The numerical mass correlating to the previous voltage
* The units that the mass is in

The voltages should be obtained using the getVoltage method in the Scales module. The numerical mass is input by the user, and the units should be specified.

### Components

There are two functions in the calibration module: calibration, and convert\_unit.

### calibration

The calibration method performs the actual calibration. It requires, as parameters, the four values specified above: unsigned integers with the voltages for 0 and some other mass, a signed integer for the actual mass, and a character of 0 for grams and 1 for ounces.  
  
The method starts by calculating the difference in voltage between the upper and lower masses. If this value is 0, the function quits, as this value would result in a divide-by-zero error.  
  
The function then proceeds to calculate the calibration parameters as specified in the Conceptual Design: Calibration section. Specifically, the following equations are used:

The second equation was, for technical reasons, split into two parts: the first calculating the value, and the second applying the negative sign.

Finally, the results are converted to the non-active unit (gram -> ounces or ounces -> grams) and all the results are stored in memory.

### convert\_unit

The convert\_unit function is simply designed to convert a mass from grams to ounces, or ounces to grams. It uses a lookup table of conversion factors to multiply the input mass. Float arithmetic is used to assure maximum accuracy.